School of Computing and Information Systems
The University of Melbourne
COMP90049
Knowledge Technologies (Semester 1, 2019)
Workshop sample solutions: Week 7

1. What are the four primary components of a **Web–scale Information Retrieval engine**? Briefly describe our goal in each of them.

   - **Crawling**: finding and downloading as many documents as we can from the web (hopefully all of them, although this isn't possible in practice).
   - **Parsing**: turning each document into a list of tokens (or terms), probably by removing page metadata, case folding, stemming, etc.
   - **Indexing**: building an inverted index out of all of the tokens in our downloaded document collection. (We stop worrying about the original documents at this point.)
   - **Querying**: after the previous three steps have been completed (off-line), we are ready to accept user queries (on-line), where we aim to produce a document ranking. The queries take the form of keywords, that we tokenise (in a similar manner to our document collection) and then apply our querying model (e.g. TF-IDF) based on the information in the inverted index.
   - (Optionally) **Additional things**: change the above ranking, based on ad-hoc application of certain factors, for example, PageRank, HITS, click-through data, zones, anchor text, etc.

2. Recall the (hypothetical) method of **crawling** given in the lectures:

   (a) Would this method be *effective* at solving the problem of crawling? Why or why not?

   - Somewhat, although it depends on having a large, random set of seeds, which isn't really possible in practice. The method will miss large numbers of pages that aren't linked to from pages in the "core" of the World Wide Web, as well as all sorts of rich data encoded in databases, etc.

   (b) Would this method be *efficient* at solving the problem of crawling? Why or why not?

   - An efficient approach depends on having a good model of **web page duplication**, so that we can decide (quickly) whether a given page has already been crawled. For example, having a hash function with respect to the page's contents (although consider how large the set is, and how difficult it would be to avoid collisions!).

3. When we **tokenise** text, we often **canonicalise** it. What are these generally accepted as referring to?
   (Note the terminology is not used consistently in the literature; for example "tokenisation" occasionally refers to all three ideas.)

   - Tokenisation, here, means decomposing the larger document into smaller information–bearing units ("tokens") than we can compare against (the keywords present in) our query.
   - Canonicalisation, here, means having a single representation of (each token within) a text, which we can use to sensibly compare one text with another (in our case, a document on the Web, and a query).

   (a) What are some issues that arise when canonicalising text written in English?

   - There are various point suggested in the lectures, including:
     - Stopwords
     - Stemming
     - Date/number formatting
     - Dialect variation
     - Spelling errors

    - Hyphenated tokens
    - Compound words
    - The genitive *'s*
  - Note that English is **easy** compared to many languages!

(b) (EXTENSION) What are some issues that might arise when canonicaling text written in other languages?

4. Assume that we have crawled the following "documents":

    (1) The South Australian Tourism Commission has defended a marketing strategy which pays celebrities to promote Kangaroo Island tourism to their followers on Twitter.
    (2) Mr O'Loughlin welcomed the attention the use of Twitter had now attracted.
    (3) Some of the tweeting refers to a current television advertisement promoting Kangaroo Island.
    (4) Those used by the Commission have included chef Matt Moran, TV performer Sophie Falkiner and singer Shannon Noll.
    (5) He said there was nothing secretive about the payments to celebrities to tweet the virtues of a tourism destination.
    (6) Marketing director of SA Tourism, David O'Loughlin, said there was no ethical problem with using such marketing and it might continue to be used.
    (7) Depending on their following, celebrities can be paid up to $750 for one tweet about the island.

- Parse each document into terms.
  - Let's consider a term to be a token with whitespace or punctuation etc. (\b) on either side. Let's also strip punctuation and fold case, and apply stemming (so that `Australian` and `australia` are instances of the same token). We might also remove stop words, although were asked to index some of them below.
- Construct an inverted index over the documents, for (at least) the terms `and`, `australia`, `celebrity`, `commission`, `island`, `on`, `the`, `to`, `tweet`, `twitter`
  - Assuming we've stemmed the tokens in the document collection, an inverted index for these terms (with term frequencies in parentheses) might look like the following:

`and`	→	4(1)	→	6(1)								
`australia`	→	1(1)										
`celebrity`	→	1(1)	→	5(1)	→	7(1)						
`commission`	→	1(1)	→	4(1)								
`island`	→	1(1)	→	3(1)	→	7(1)						
`on`	→	1(1)	→	7(1)								
`the`	→	1(1)	→	2(2)	→	3(1)	→	4(1)	→	5(2)	→	7(1)
`to`	→	1(2)	→	3(1)	→	5(2)	→	6(1)	→	7(1)		
`tweet`	→	3(1)	→	5(1)	→	7(1)						
`twitter`	→	1(1)	→	2(1)								

- Using the vector space model and the cosine measure, rank the documents for the query `commission to island on twitter`

  (a) Using the weighting functions $w_{d,t} = f_{d,t}$ and $w_{q,t} = \frac{N}{f_t}$
    - For the query `commission to island on twitter`, we take each term and find its weight in the query. To do this, say for `commission`, we observe that there are 7

documents in the collection, and 2 of them contain `commission`, and so on:

$$\begin{aligned}
w_{q,\texttt{commission}} &= \frac{N}{f_{\texttt{commission}}} \\
&= \frac{7}{2} = 3.500 \\
w_{q,\texttt{island}} &= \frac{7}{3} \approx 2.333 \\
w_{q,\texttt{on}} &= \frac{7}{2} = 3.500 \\
w_{q,\texttt{to}} &= \frac{7}{5} = 1.400 \\
w_{q,\texttt{twitter}} &= \frac{7}{2} = 3.500
\end{aligned}$$

– All of the other term weights in the query are 0, so we can ignore them (and the weight documents are unimportant for the dot-product). The document weights are just the frequency of the term in each document (which we have recorded in our inverted index above). For example, for `island` our representation would look like:

$$\begin{aligned}
w_{d_1,i} &= 1 \\
w_{d_2,i} &= 0 \\
w_{d_3,i} &= 1 \\
w_{d_4,i} &= 0 \\
w_{d_5,i} &= 0 \\
w_{d_6,i} &= 0 \\
w_{d_7,i} &= 1
\end{aligned}$$

– The cosine model gives us:

$$\begin{aligned}
\cos(d_1, q) &= \frac{\langle 1,1,1,2,1\rangle \cdot \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle}{\mid \langle 0,1,1,1,1,1,1,2,0,1\rangle \mid \cdot \mid \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle \mid} \\
&\approx \frac{15.633}{\sqrt{11}\sqrt{44.15}} \approx 0.709 \\
\cos(d_2, q) &= \frac{\langle 0,0,0,0,1\rangle \cdot \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle}{\mid \langle 0,0,0,0,0,0,2,0,0,1\rangle \mid \cdot \mid \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle \mid} \\
&\approx \frac{3.5}{\sqrt{5}\sqrt{44.15}} \approx 0.236 \\
\cos(d_3, q) &= \frac{\langle 0,1,0,1,0\rangle \cdot \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle}{\mid \langle 0,0,0,0,1,0,1,1,1,0\rangle \mid \cdot \mid \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle \mid} \\
&\approx \frac{3.733}{\sqrt{4}\sqrt{44.15}} \approx 0.281 \\
\cos(d_4, q) &= \frac{\langle 1,0,0,0,0\rangle \cdot \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle}{\mid \langle 1,0,0,1,0,0,1,0,0,0\rangle \mid \cdot \mid \langle 3.5, 2.333, 3.5, 1.4, 3.5\rangle \mid} \\
&\approx \frac{3.5}{\sqrt{3}\sqrt{44.15}} \approx 0.304
\end{aligned}$$

$$\begin{aligned}
\cos(d_5,q) &= \frac{\langle 0,0,0,2,0\rangle \cdot \langle 3.5,2.333,3.5,1.4,3.5\rangle}{\mid \langle 0,0,1,0,0,0,2,2,1,0\rangle \mid \cdot \mid \langle 3.5,2.333,3.5,1.4,3.5\rangle \mid} \\
&\approx \frac{2.8}{\sqrt{10}\sqrt{44.15}} \approx 0.133 \\
\cos(d_6,q) &= \frac{\langle 0,0,0,1,0\rangle \cdot \langle 3.5,2.333,3.5,1.4,3.5\rangle}{\mid \langle 1,0,0,0,0,0,0,1,0,0\rangle \mid \cdot \mid \langle 3.5,2.333,3.5,1.4,3.5\rangle \mid} \\
&\approx \frac{1.4}{\sqrt{2}\sqrt{44.15}} \approx 0.149 \\
\cos(d_7,q) &= \frac{\langle 0,1,1,1,0\rangle \cdot \langle 3.5,2.333,3.5,1.4,3.5\rangle}{\mid \langle 0,0,1,0,1,1,1,1,1,0\rangle \mid \cdot \mid \langle 3.5,2.333,3.5,1.4,3.5\rangle \mid} \\
&\approx \frac{7.233}{\sqrt{6}\sqrt{44.15}} \approx 0.444
\end{aligned}$$

- All of the documents have non-zero similarity, so they all get returned, and the order is: {1,7,4,3,2,6,5}.
- Notice also that each cosine calculation involves the same division by the length of the query (in this case, $\sqrt{44.15}$) — since all we care about is the order of the documents and not the actual score, we can safely leave this term out (but it isn't the cosine anymore, so I'll call it $\cos'$ below).

(b) Using the weighting functions $w_{d,t} = 1 + \log_2 f_{d,t}$ and $w_{q,t} = \log_2(1 + \frac{N}{f_t})$

- Let's observe first that nothing has changed with the document weights: frequencies of 0 still have a weight of 0; frequencies of 1 are now $1+\log_2(1) = 1+0 = 1$; frequencies of 2 are now $1+\log_2(2) = 1+1 = 2$. (Higher frequency terms would have their weights reduced with this model.) This also means that the document lengths are the same.
- We can now calculate the query weights according to the new model:
  q: $\langle 2.170, 1.737, 2.170, 1.263, 2.170\rangle$, and the cosine calculations look similar:

$$\begin{aligned}
\cos'(d_1,q) &= \frac{\langle 1,1,1,2,1\rangle \cdot \langle 2.170,1.737,2.170,1.263,2.170\rangle}{\mid \langle 0,1,1,1,1,1,1,2,0,1\rangle \mid} \\
&\approx \frac{10.773}{\sqrt{11}} \approx 3.248 \\
\cos'(d_2,q) &= \frac{\langle 0,0,0,0,1\rangle \cdot \langle 2.170,1.737,2.170,1.263,2.170\rangle}{\mid \langle 0,0,0,0,0,0,2,0,0,1\rangle \mid} \\
&\approx \frac{2.170}{\sqrt{5}} \approx 0.970 \\
\cos'(d_3,q) &= \frac{\langle 0,1,0,1,0\rangle \cdot \langle 2.170,1.737,2.170,1.263,2.170\rangle}{\mid \langle 0,0,0,0,1,0,1,1,1,0\rangle \mid} \\
&\approx \frac{3.000}{\sqrt{4}} \approx 1.500 \\
\cos'(d_4,q) &= \frac{\langle 1,0,0,0,0\rangle \cdot \langle 2.170,1.737,2.170,1.263,2.170\rangle}{\mid \langle 1,0,0,1,0,0,1,0,0,0\rangle \mid} \\
&\approx \frac{2.170}{\sqrt{3}} \approx 1.253 \\
\cos'(d_5,q) &= \frac{\langle 0,0,0,2,0\rangle \cdot \langle 2.170,1.737,2.170,1.263,2.170\rangle}{\mid \langle 0,0,1,0,0,0,2,2,1,0\rangle \mid} \\
&\approx \frac{2.526}{\sqrt{10}} \approx 0.799
\end{aligned}$$

$$\begin{aligned}\cos'(d_6,q) &= \frac{\langle 0,0,0,1,0\rangle \cdot \langle 2.170, 1.737, 2.170, 1.263, 2.170\rangle}{\mid \langle 1,0,0,0,0,0,0,1,0,0\rangle \mid} \\ &\approx \frac{1.263}{\sqrt{2}} \approx 0.893 \\ \cos'(d_7,q) &= \frac{\langle 0,1,1,1,0\rangle \cdot \langle 2.170, 1.737, 2.170, 1.263, 2.170\rangle}{\mid \langle 0,0,1,0,1,1,1,1,1,0\rangle \mid} \\ &\approx \frac{5.170}{\sqrt{6}} \approx 2.111\end{aligned}$$

  - The document ranking has changed a little bit this time (because the relative differences between common terms and rare terms is smaller due to the log): {1,7,3,4,2,6,5}
  - Also note that some of the values are greater than 1, because I left out the division by the length of the query (*cos′*), which gives a factor of $\sqrt{18.74}$ in the denominator — it doesn't change the order though.

5. When querying, what is an **accumulator**? What is the main problem, if we wish to use them? What heuristics can we use to solve this problem?

   - We use accumulators to keep track of the partial sum of the dot product (for the cosine similarity) as we process query terms one-by-one. This makes sense, because our inverted index keeps track of the documents where a given term appears: consequently, we can take a term from the query, read off each document from the index, and update (add) the TF-IDF value for that document to the corresponding accumulator. Once we have processed all of the query terms, we divide each accumulator by its corresponding document length, and then take the greatest values as the top results.
   - The list of accumulators is typically an array (we need random-access when writing to make it worth our while in terms of time). We need a single accumulator for each document — unfortunately, there are **lots** of documents. A collection with 10M documents, using 32-bit floats for each accumulator value, would require 80MB of memory: not a problem, but we're going to have plenty of cache misses. For a more realistic number of documents on the Web, we have no hope of fitting this data structure into memory.
   - To solve this problem, we generally use only a few accumulators (perhaps a few hundred or a few thousand, depending on how many results we wish to return), and agressively prune the possible result space — note that most documents have a trivially low similarity anyway! There are two main pruning strategies that we use:
     - Limiting: Only create an accumulator for the first few documents; these documents are then the results set, and the further calculations are just for re-ordering. This depends on **processing the most important terms first**, namely, the ones with highest IDF: we wish to have accumulators for the documents that contain the important terms; if we run out of accumulators for documents which contain the less important terms (but not the important ones), we accept that they probably weren't going to be returned at the top of the ranking anyway.
     - Thresholding: Only create an accumulator when the TF-IDF value (for the query term we're processing) is above some threshold. Similar to the above, but documents with low-frequency rare terms might not get added to the result set; whereas documents with high-frequency moderately rare terms might get added instead.